Session Management with SessionAuth

From Simple Concepts to Working Code

Sessions

- 🤔 What are sessions and why do we need them?
- 🔧 How PHP sessions work step-by-step
- 💻 Building a simple session system
- 🛡️ Session security best practices
- 🎯 Hands-on coding examples

The Problem: HTTP is Stateless

```
User visits page 1: "Hi, I'm John"
User visits page 2: "Who are you again?" 🤦
```

Each request is independent

- Server forgets everything after each request
- No memory of previous interactions
- How do we remember users across pages?

Solution: Sessions!

What Are Sessions?

Sessions = Server's memory of your visit

```php
// When you login:
$_SESSION['username'] = 'john';
$_SESSION['logged_in'] = true;

// Later, on any page:
echo "Welcome back, " . $_SESSION['username'];
// Outputs: Welcome back, john
```

Key Concept

- Server stores your data
- Browser gets a "ticket number" (session ID)
- Browser shows ticket, server finds your data

How Sessions Work: Step by Step

Step 1: Start Session

```
session_start(); // Must be first line!
```

Step 2: Store Data

```
$_SESSION['user_id'] = 123;
$_SESSION['username'] = 'john';
```

Step 3: PHP Magic

- Creates unique session ID: `abc123def456`
- Sends cookie to browser: `PHPSESSID=abc123def456`
- Saves your data on the server

Step 4: Remember Later

- Browser sends cookie with every request
- PHP loads your data automatically
- `$_SESSION` array is ready to use!

Session PHP Example

login.php (The Core Idea)

In this PHP program:

```php
<?php
session_start();

if (!$_SESSION['logged_in']) {
    header('Location: login.php');
    exit;
}
?>

<h1>Welcome!</h1>
<p>Hello, <?= $_SESSION['username'] ?>!</p>
<p>Your user ID is: <?= $_SESSION['user_id'] ?></p>
<p>You are logged in.</p>
<p><a href="logout.php">Logout</a></p>
<p><a href="check.php">Check Session</a> | <a href="basic.php">Basic Example</a></p>
```

- The PHP server **executes** the PHP code and **outputs** HTML (or other content) as a result. Only the output is sent to the browser.
  - The `<?php ... ?>` tags enclose PHP code that is executed **on the server**.
  - PHP code can be embedded into an HTML document, allowing dynamic content (like user data from sessions) to be inserted into the page sent to the browser.
- When a user requests this `.php` file through a web server (like using `php -S localhost:8000`), the **PHP interpreter processes any code inside `<?php ?>`**.
  - The output of the PHP code (usually HTML or text) is sent to the **client's browser**, which only sees the final HTML—not the PHP code.

- Users can make a POST request using this form with username and password.

```
<form method="post">
 Username: <input type="text" name="username"><br>
 Password: <input type="password" name="password"><br>
    <input type="submit" value="Login">
</form>
```

- In this example, the `session_start()` function resumes the user session, and if the `$_SESSION['logged_in']` value is false or unset, the user is redirected to `login.php`.
- If the session is valid, PHP dynamically inserts the username and user ID into the HTML.

```php
<?php
session_start();
if ($_POST['username'] == 'john' && $_POST['password'] == 'secret') {
    // SUCCESS: Store user in session
    $_SESSION['username'] = 'john';
    $_SESSION['logged_in'] = true;

    header('Location: welcome.php');
    exit;
}
?>
```

welcome.php (Session Protection)

```php
<?php
session_start();

// Check if user is logged in
if (!$_SESSION['logged_in']) {
    header('Location: login.php');  // Redirect to login
    exit;
}
?>

<h1>Welcome!</h1>
<p>Hello, <?= $_SESSION['username'] ?>!</p>
<p>You are logged in.</p>

<a href="logout.php">Logout</a>
```

Magic: This page "knows" who you are!

logout.php (Destroy Session)

```php
<?php
session_start();
session_destroy();  // Forget everything
?>

<h1>Logged Out</h1>
<p>Your session has been destroyed.</p>
<a href="login.php">Login Again</a>
```

What happens

- `session_destroy()` deletes all session data
- Server forgets who you are
- Protected pages will redirect to login

check.php (See What's Stored)

```php
<?php
session_start();
?>

<h1>Session Info</h1>

<p><strong>Session ID:</strong> <?= session_id() ?></p>

<p><strong>What's in $_SESSION:</strong></p>
<pre><?php print_r($_SESSION); ?></pre>

<?php if ($_SESSION['logged_in']): ?>
 <p>Status: ✅ Logged in as <?= $_SESSION['username'] ?></p>
<?php else: ?>
 <p>Status: ❌ Not logged in</p>
<?php endif; ?>
```

Rewrite using the SessionAuth class

Problems with Basic Approach

Code Duplication 🔄

```
// Must copy this to every protected page
session_start();
if (!$_SESSION['logged_in']) {
    header('Location: login.php');
    exit;
}
```

Security Issues 🚨

- No session regeneration (session fixation attacks)
- Incomplete logout (session data may remain)
- Inconsistent authentication checks

Solution: SessionAuth Class

Object-Oriented Approach = Better Code + Better Security

```php
class SessionAuth {
    public function __construct() {
        if (session_status() === PHP_SESSION_NONE) {
            session_start();
        }
    }

    public function login_user($user) {
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['username'] = $user['username'];
        $_SESSION['logged_in'] = true;

        // Security: Prevent session fixation
        session_regenerate_id(true);
    }
```

```php
    public function logout_user() {
        session_unset();   // Clear data
        session_destroy(); // Destroy session
    }

    public function is_logged_in() {
        return isset($_SESSION['logged_in']) && $_SESSION['logged_in'] === true;
    }
```

- These functions require the user to be logged in.

```
    public function get_user() {
        if ($this->is_logged_in()) {
            return [
                'id' => $_SESSION['user_id'],
                'username' => $_SESSION['username']
            ];
        }
        return null;
    }

    public function require_auth($login_url = 'login.php') {
        if (!$this->is_logged_in()) {
            header("Location: $login_url");
            exit;
        }
    }
}
```

SessionAuth Benefits

🎯 Clean Code

```
// Old way (repeated everywhere)
session_start();
if (!$_SESSION['logged_in']) {
    header('Location: login.php');
    exit;
}

// New way (one line)
$auth = new SessionAuth();
$auth->require_auth();
```

🛡️ Built-in Security

- Automatic session regeneration on login
- Complete session cleanup on logout
- Consistent authentication across all pages

Code Comparison: Login

Basic Approach (Security Risk!)

```
if ($username === 'john' && $password === 'secret') {
    $_SESSION['user_id'] = 1;
    $_SESSION['username'] = 'john';
    $_SESSION['logged_in'] = true;
    // No session regeneration! 🚨
}
```

Session Fixation Attack

Problem: Attacker sets victim's session ID, waits for login

```
// Basic approach - VULNERABLE
$_SESSION['logged_in'] = true;
// Session ID stays the same! 🚨
```

Solution: SessionAuth regenerates the session ID

SessionAuth Approach (Secure!)

```php
if ($username === 'john' && $password === 'secret') {
    $user_data = ['id' => 1, 'username' => 'john'];
    $auth->login_user($user_data); // Includes security features ✅
}
```

SessionAuth automatically prevents session fixation attacks!

Code Comparison: Authentication

Basic Approach (Must Repeat)

```
// Copy this to EVERY protected page
session_start();
if (!$_SESSION['logged_in']) {
    header('Location: login.php');
    exit;
}
// Easy to forget! 😰
```

SessionAuth Approach (DRY Principle)

```
// One line protects any page
$auth = new SessionAuth();
$auth->require_auth(); // Clean and consistent! 😊
```

DRY = Don't Repeat Yourself

Hands-On Exercise

Try Both Approaches

Basic Version: `/session/basic/`

- Simple but has security issues
- Code duplication everywhere
- Manual session management

SessionAuth Version: `/session/sessionauth/`

- Secure and professional
- Clean, reusable code
- Automatic security features

Other Session Related Topics

Session vs Other Storage

Method	Server Storage	Client Storage	Security	Expiration
Sessions	✅ Yes	Session ID only	✅ High	Server controls
Cookies	❌ No	✅ Full data	⚠️ Medium	Client controls
LocalStorage	❌ No	✅ Full data	❌ Low	Manual only
JWT Tokens	❌ No	✅ Encoded data	⚠️ Medium	Token controls

Sessions are most secure for sensitive data!

What you could add to SessionAuth

```php
// Password hashing
public function verify_password($input, $hash) {
    return password_verify($input, $hash);
}

// Session timeout
public function check_timeout($minutes = 30) {
    // Implementation here
}
```

```php
// Remember me functionality
public function set_remember_token($user_id) {
    // Implementation here
}

// Role-based access
public function require_role($role) {
    // Implementation here
}
```

Common Session Problems & Solutions

Problem 1: Session Not Starting

```php
// ❌ Wrong - session already started elsewhere
session_start();

// ✅ Correct - check first
if (session_status() === PHP_SESSION_NONE) {
    session_start();
}
```

Problem 2: Headers Already Sent

```php
// ❌ Wrong - output before session_start()
echo "Hello";
session_start(); // Error!

// ✅ Correct - session_start() first
session_start();
echo "Hello";
```

Problem 3: Session Data Lost

```php
// ❌ Wrong - forgot to start session
$_SESSION['user'] = 'john'; // Won't work!

// ✅ Correct - start session first
session_start();
$_SESSION['user'] = 'john';
```

Session Best Practices

1. Always Check Session Status

```php
public function __construct() {
    if (session_status() === PHP_SESSION_NONE) {
        session_start();
    }
}
```

2. Secure Session Configuration

```php
// In your main configuration file
ini_set('session.cookie_httponly', 1); // Prevent JavaScript access
ini_set('session.cookie_secure', 1);   // HTTPS only
ini_set('session.use_strict_mode', 1); // Strict session ID generation
```

3. Clean Session Data

```php
public function logout_user() {
    session_unset();   // Clear all $_SESSION data
    session_destroy(); // Destroy the session file

    // Optional: Clear the session cookie
    if (isset($_COOKIE[session_name()])) {
        setcookie(session_name(), '', time() - 3600, '/');
    }
}
```

Real-World Applications

E-commerce Sites

- Shopping cart contents
- User preferences
- Login status

Social Media

- User identity
- Privacy settings
- Recent activity

Banking Applications

- Secure authentication
- Transaction state
- Security timeouts

Your Projects

- User management systems
- Admin panels
- Any application requiring login

Key Takeaways

Sessions Enable

1. **User Authentication** - Remember who's logged in
2. **State Management** - Maintain data across requests
3. **Security** - Server-side data storage
4. **User Experience** - No need to login repeatedly

Security Features

- **Session ID regeneration** prevents fixation attacks
- **Timeout handling** limits exposure window
- **Activity tracking** enables idle detection
- **Proper cleanup** prevents data leaks